Set-4

1. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Develop a lexical Analyzer to identify identifiers, constants, operators using C program.

Code:

```
%{
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define MAX_ID_LEN 31

int char_count = 0, line_count = 0, word_count = 0;

void check_identifier(const char *yytext) {
  if (strlen(yytext) > MAX_ID_LEN) {
    printf("Warning: Identifier '%s' exceeds maximum length and is
truncated.\n", yytext);
  }
}
%}

%%
// Ignore spaces, tabs, and new lines
[ \t]+   ;
\n     { line_count++; char_count++; }

// Ignore single-line comments
//.*     ;

// Ignore multi-line comments
/\*([^*]|\*+[^/*])*\*+/ ;

// Identify identifiers
[A-Za-z_][A-Za-z0-9_]* { check_identifier(yytext); word_count++; char_count
+= yyleng; printf("Identifier: %s\n", yytext); }

// Identify constants (integer and floating point numbers)
```

```
[0-9]+\.?[0-9]* { word_count++; char_count += yyleng; printf("Constant:
%s\n", yytext); }

// Identify operators
[+\-*/=<>!&|%^]+ { char_count += yyleng; printf("Operator: %s\n", yytext); }

// Count other characters
.     { char_count++; }

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  printf("Characters: %d\n", char_count);
  printf("Lines: %d\n", line_count);
  printf("Words: %d\n", word_count);
  return 0;
}
```

2. Design a lexical Analyzer to validate operators to recognize the operators +,-,*,/ using regular Arithmetic operators .

Code:

```
%{
#include <stdio.h>
int char_count = 0, line_count = 0, operator_count = 0;
%}

%%
// Ignore spaces, tabs, and new lines
[ \t]+   ;
\n     { line_count++; char_count++; }

// Ignore single-line comments
```

```
//.*     ;

// Ignore multi-line comments
/\*([^*]|\*+[^/*])*\*+/ ;

// Recognize arithmetic operators
[+\-*/] { operator_count++; char_count += yyleng; printf("Operator: %s\n",
yytext); }

// Count other characters
.     { char_count++; }

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  printf("Characters: %d\n", char_count);
  printf("Lines: %d\n", line_count);
  printf("Operators: %d\n", operator_count);
  return 0;
}
```

3. Write a LEX program to check whether the given input is digit or not.

Code:

```
%{
#include <stdio.h>
int digit_count = 0, non_digit_count = 0;
%}

%%
// Ignore spaces, tabs, and new lines
[ \t\n]+   ;

// Recognize digits
```

```
[0-9]+ { digit_count++; printf("Digit: %s\n", yytext); }

// Recognize non-digit characters
[^0-9] { non_digit_count++; printf("Non-digit: %s\n", yytext); }

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  printf("Total Digits: %d\n", digit_count);
  printf("Total Non-Digits: %d\n", non_digit_count);
  return 0;
}
```

4. Implement Lexical Analyzer using LEX or FLEX (Fast Lexical Analyzer). The program should separate the tokens in the given C program and display with appropriate caption.

Input Source Program: (sample.c)

```
#include<stdio.h>
void main()
{
int a,b,c = 30;
printf("hello");
}
```

Code:

```
%{
#include <stdio.h>
#include <string.h>
%}

%%
// Ignore spaces, tabs, and new lines
[ \t]+   ;
```

```
\n      { printf("New Line\n"); }

// Keywords
"int"|"void"|"return" { printf("Keyword: %s\n", yytext); }

// Identifiers
[A-Za-z_][A-Za-z0-9_]* { printf("Identifier: %s\n", yytext); }

// Constants
[0-9]+ { printf("Constant: %s\n", yytext); }

// String literals
".*" { printf("String Literal: %s\n", yytext); }

// Operators
[+\-*/=] { printf("Operator: %s\n", yytext); }

// Delimiters
[;(){}] { printf("Delimiter: %s\n", yytext); }

// Preprocessor directives
#[a-zA-Z]+ { printf("Preprocessor Directive: %s\n", yytext); }

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  return 0;
}
```